## 1. PROPOSITO

Emitir los lineamientos generales para el análisis, diseño e implementación de la arquitectura de software para el desarrollo de aplicaciones.

## 2. ALCANCE

El presente documento permite definir la arquitectura de software de N-Capas en cualquier proyecto de desarrollo de software que inicie en la entidad y se ha desarrollado con las herramientas y lenguaje de programación de C# .NET por medio de DDD (Domain Driven Desing). Este documento solo pretende usar una arquitectura dentro muchas y con herramientas de Microsoft

## 3. GLOSARIO

- **Patrones de software:** son formas de capturar estructuras de diseño de probada eficacia, para que puedan ser reutilizadas.

- **Arquitectura de Software:** El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad, disponibilidad, eficiencia o usabilidad.

- **Arquitectura Microkernel:** Se aplica a los sistemas de software que deben ser capaces de adaptarse a los requisitos cambiantes del sistema. Separa un núcleo funcional mínimo de la funcionalidad ampliada y de las partes específicas del cliente. También sirve de enchufe para conectar estas extensiones y coordinar su colaboración

- **Arquitetura Microservicios:** es un enfoque para desarrollar aplicaciones de servidores como una serie de pequeños servicios, comúnmente es orientado al Back-end, pero también esta siendo orientado al front-end

- **Arquitectura en Capas:** Las Capas (Layers) se ocupan de la división lógica de componentes y funcionalidad, y no tienen en cuenta la localización física de componentes en diferentes servidores o en diferentes lugares.

- **Software:** Serie de instrucciones y datos, que permiten aprovechar todos los recursos que el computador tiene, de manera que pueda resolver gran cantidad de problemas.

- **Arquitectura Basadas en Evento:** arquitectura asíncrona distribuida para desarrollar un sistema altamente escalable. La arquitectura consiste en componentes de procesamiento de eventos de un solo propósito que escuchan los eventos y los procesan asincrónicamente. La arquitectura impulsada por eventos construye una unidad central que acepta todos los datos y luego los delega a los módulos separados que manejan el tipo particular.

- **Arquitectura Basada en el Espacio:** diseñado específicamente para abordar y resolver problemas de escalabilidad y concurrencia. También es un patrón de arquitectura útil para las aplicaciones que tienen volúmenes de usuarios concurrentes variables e impredecibles. La alta escalabilidad se logra eliminando la restricción de la base de datos central y utilizando en su lugar cuadrículas de datos replicados en memoria.

## 4. RESPONSABLE

Gestor de desarrollo
Jefe oficina asesora de informática

## 5. INSTRUCCIONES:

El tipo de arquitectura a implementar depende del tipo de aplicación que se quiere desarrollar e implementar. Todas las decisiones sobre arquitectura deben plasmarse en una documentación que sea entendible por todos los integrantes del equipo de desarrollo, así como por el resto de los participantes del proyecto, incluidos los clientes. Para describir la arquitectura se optó por metodologías gráficas como 4+1. 4+1 describe una arquitectura software mediante 4 vistas distintas del sistema:

- **Vista lógica:** La vista lógica del sistema muestra la funcionalidad que el sistema proporciona a los usuarios finales. Emplea para ello diagramas de clases, de comunicación y de secuencia.
- **Vista del proceso**: La vista del proceso del sistema muestra cómo se comporta el sistema tiempo de ejecución, qué procesos hay activos y cómo se comunican. La vista del proceso resuelve cuestiones como la concurrencia, la escalabilidad, el rendimiento, y en general cualquier característica dinámica del sistema.
- **Vista física**: La vista física del sistema muestra cómo se distribuyen los distintos componentes software del sistema en los distintos nodos físicos de la infraestructura y cómo se comunican unos con otros. Emplea para ello los diagramas de despliegue.
- **Vista de desarrollo**: La vista lógica del sistema muestra el sistema desde el punto de vista de los programadores y se centra en el mantenimiento del software. Emplea para ello diagramas de componentes y de paquetes. 18 Guía de Arquitectura N-Capas Orientada al Dominio
- **Escenarios:** La vista de escenarios completa la descripción de la arquitectura. Los escenarios describen secuencias de interacciones entre objetos y procesos y son usados para identificar los elementos arquitecturales y validar el diseño.

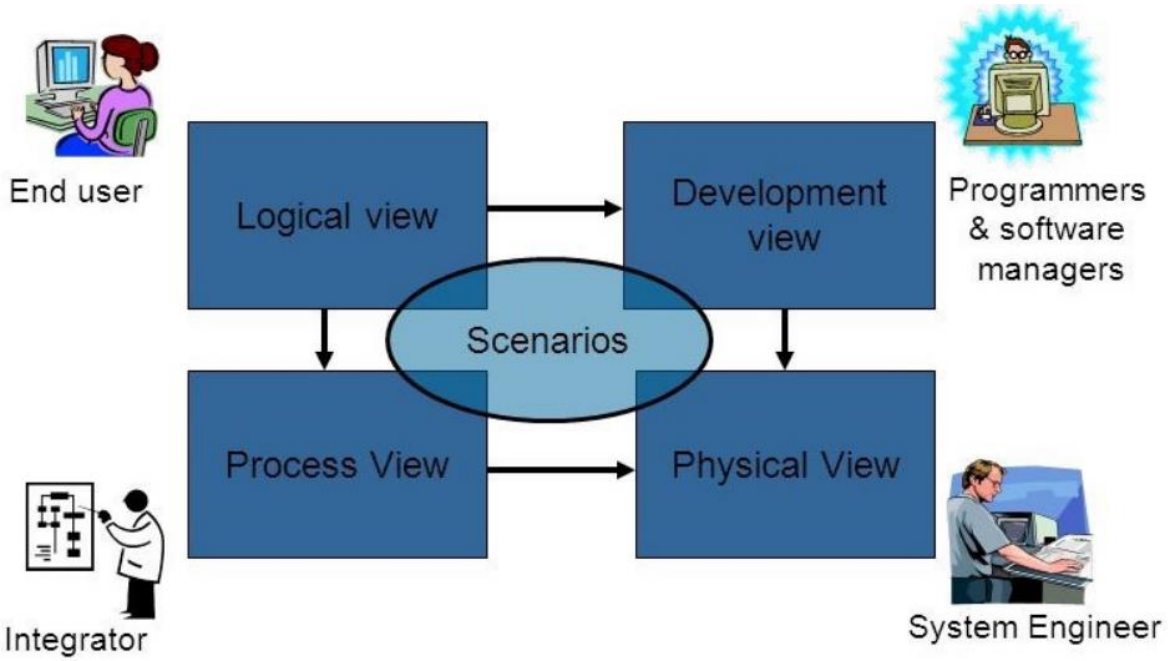



*Ilustración 1 Vista-Modelo 4+1 Arquitectura de Software*

Las capas son agrupaciones horizontales lógicas de componentes de software que forman la aplicación o el servicio. Nos ayudan a diferenciar entre los diferentes tipos de tareas a ser realizadas por los componentes, ofreciendo un diseño que maximiza la reutilización y, especialmente, la mantenibilidad. En definitiva, se trata de aplicar el principio de „*Separación de Responsabilidades*" (SoC - *Separation of Concerns principle*) dentro de una Arquitectura. Cada capa lógica de primer nivel puede tener un número concreto de componentes agrupados en sub-capas. Dichas sub-capas realizan a su vez un tipo específico de tareas. Al identificar tipos genéricos de componentes que existen en la mayoría de las soluciones, podemos construir un patrón o mapa de una aplicación o servicio y usar dicho mapa como modelo de nuestro diseño. El dividir una aplicación en capas separadas que desempeñan diferentes roles y funcionalidades, nos ayuda a mejorar el mantenimiento del código; nos permite también diferentes tipos de despliegue y, sobre todo, nos proporciona una clara 24 Guía de Arquitectura N-Capas Orientada al Dominio delimitación y situación de dónde debe estar cada tipo de componente funcional e incluso cada tipo de tecnología.

## 1. Diseño básico de capas

Se deben separar los componentes de la solución en capas. Los componentes de cada capa deben ser cohesivos y tener aproximadamente el mismo nivel de abstracción. Cada capa de primer nivel debe de estar débilmente acoplada con el resto de capas de primer nivel. El proceso es como sigue: Comenzar en el nivel más bajo de abstracción, por ejemplo „Capa 1". Esta capa es la base del sistema. Se continúa esta escalera abstracta con otras capas (Capa J, Capa J-1) hasta el último nivel (Capa-N):

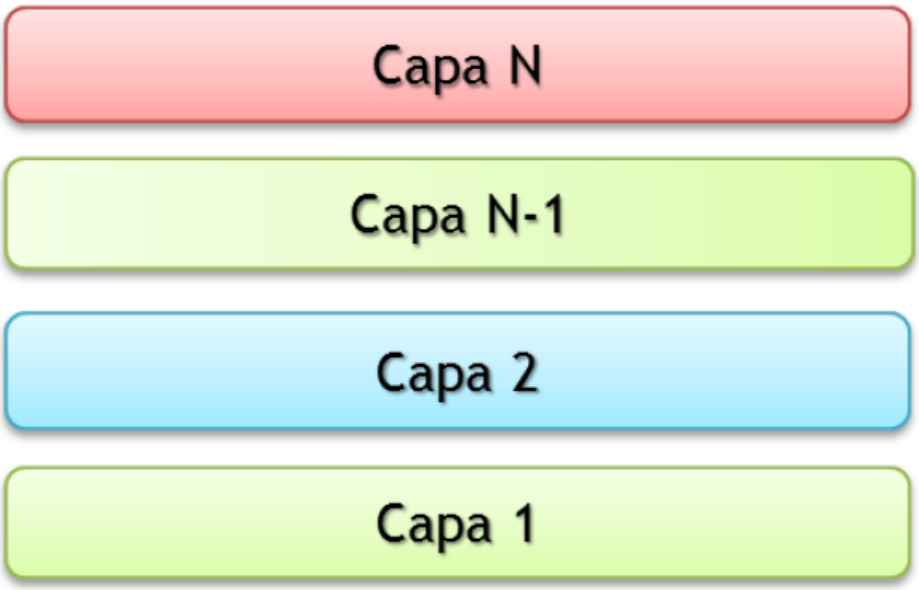



*Ilustración 2 Modelo N-Capas*

2. **Principios de Diseño "SOLID":** para el desarrollo de una aplicación, en la alcaldía de Cartagena, se debe tener en cuenta estos principios. El acrónimo deriva de las siguientes frases/principios en inglés:

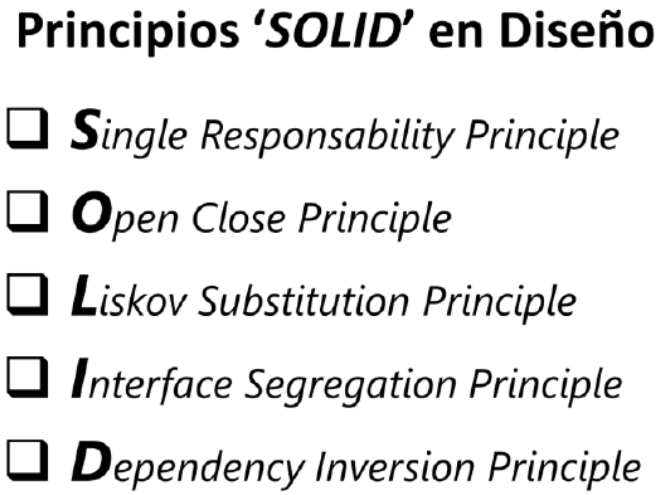



*Ilustración 3 Principios SOLID*

De una forma resumida, los principios de diseño SOLID son los siguientes:

- **Principio de Única Responsabilidad (*'Single Responsability Principle*'')**: *Una clase debe tener una única responsabilidad o característica*. Dicho de otra manera, una clase debe de tener una única razón por la que está justificado realizar cambios sobre su código fuente. Una consecuencia de este principio es que, de forma general, las clases deberían tener pocas dependencias con otras clases/tipos.
- **Principio Abierto Cerrado („*Open Closed Principle*"):** *Una clase debe estar abierta para la extensión y cerrada para la modificación.* Es decir, el comportamiento de una clase debe poder ser extendido sin necesidad de realizar modificaciones sobre el código de la misma.
- **Principio de Sustitución de Liskov („*Liskov Subtitution Principle*"):** *Los subtipos deben poder ser sustituibles por sus tipos base (interfaz o clase base).* Este hecho se deriva de que el comportamiento de un programa que trabaja con abstracciones (interfaces o clases base) no debe cambiar porque se sustituya una implementación concreta por otra. Los programas deben hacer referencia a las abstracciones, y no a las implementaciones.
- **Principio de Segregación de Interfaces („*Interface Segregation Principle*'):** *Los implementadores de Interfaces de clases no deben estar obligados a implementar métodos que no se usan*. Es decir, los interfaces de clases deben ser específicos dependiendo de quién los consume y por lo tanto, tienen que estar granularizados/segregados en diferentes interfaces no debiendo crear nunca grandes interfaces. Las clases deben exponer interfaces separados para diferentes clientes/consumidores que difieren en los requerimientos de interfaces.
- **Principio de Inversión de Dependencias („*Dependency Inversión Principle*"):** *Las abstracciones no deben depender de los detalles – Los detalles deben depender de las abstracciones*. Las dependencias directas entre clases deben ser reemplazadas por abstracciones (interfaces) para permitir diseños top-down sin requerir primero el diseño de los niveles inferiores.

3. **Capas de Presentación, Aplicación, Dominio e Infraestructura:** En el nivel más alto y abstracto, la vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas, similar al siguiente esquema (siguiendo las tendencias de Arquitectura DDD – **D**omain **D**riven **D**esing):

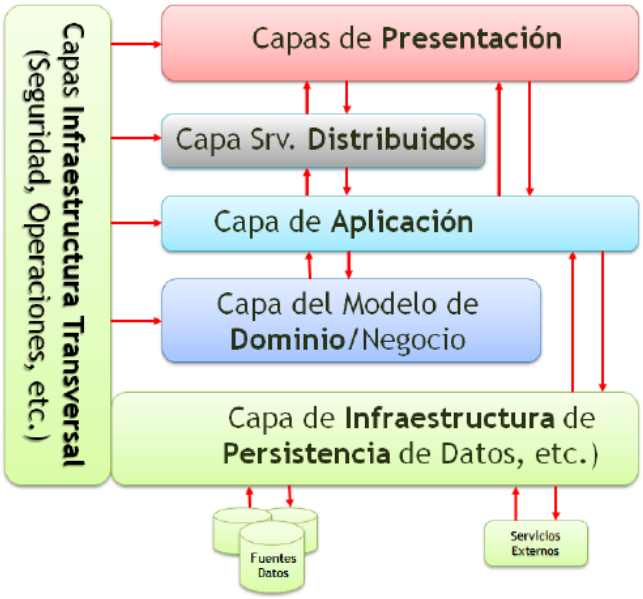



*Ilustración 4 Vista de arquitectura lógica simplificada de un sistema N-Capas DDD*

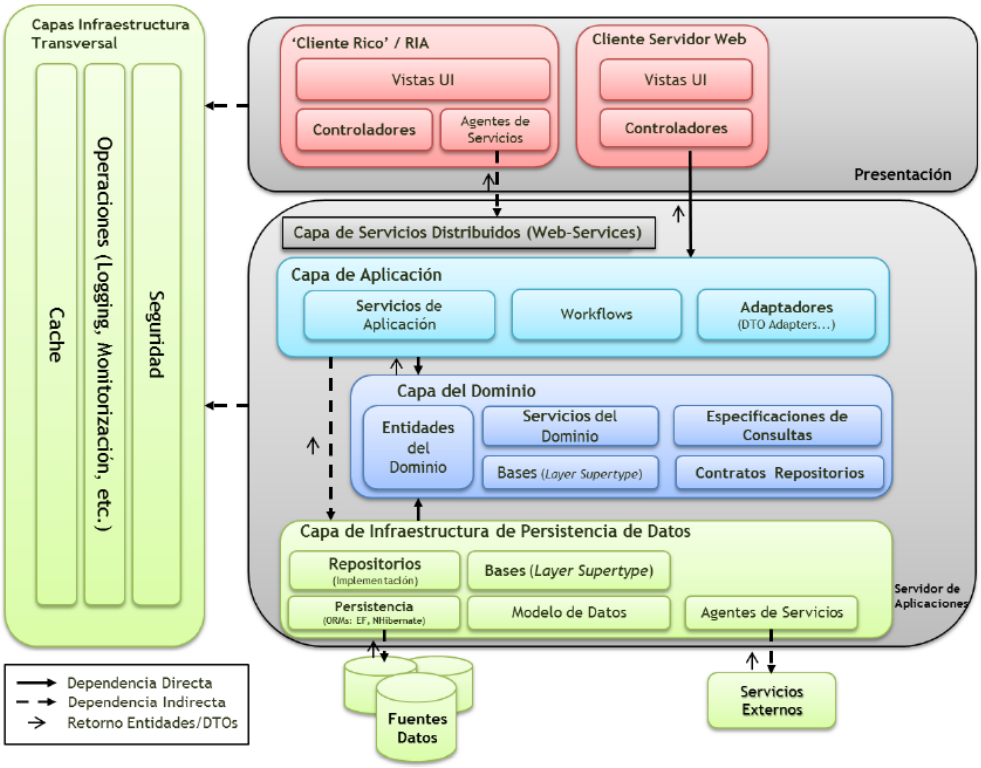
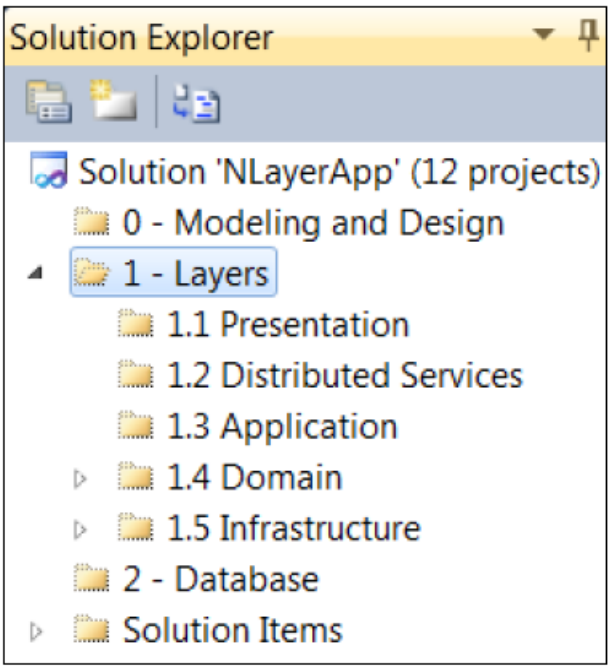


*Ilustración 5 Arquitectura N-Capas con Orientación al Dominio*

4. **Diseño de la solución de Visual Studio:** Teniendo una "Solución" de Visual Studio, inicialmente crearemos la estructura de carpetas lógicas para albergar y distribuir los diferentes proyectos. En la mayoría de los casos crearemos un proyecto (.DLL) por cada capa o sub-capa, para disponer así de una mayor flexibilidad y facilitar los posibles desacoplamientos. Sin embargo, esto ocasiona un número de proyectos considerable, por lo que es realmente imprescindible ordenarlos/jerarquizarlos mediante carpetas lógicas de Visual Studio. La jerarquía inicial sería algo similar a la siguiente:




5. **Repositorio Base Para Desarrollo De La Arquitectura De Software N-Capas con DDD:** la url para descargar el repositorio base para el desarrollo de aplicaciones en C# .NET con los componentes de DDD. Para usar el repositorio deberá solicitar al gestor de desarrollo su descarga, puede solicitarlo al correo gestordesarrollo@cartagena.gov.co

https://gestordesarrollo@dev.azure.com/gestordesarrollo/Gestion%20Informe%20de%20Actividades/_git/Gestion%20Informe%20de%20Actividades

## 6. Documentos de Referencia:

- La Guía Definitiva de Scrum: Las Reglas del Juego (https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf)
- Scrum Glossary - https://www.scrum.org/resources/scrum-glossary
- Instructivo para el desarrollo de software

## 7. CONTROL DE CAMBIOS

| VERSION | DESCRIPCIÓN DE CAMBIOS |
|---|---|
| 1.0 | Elaboración del documento |

## 8. VALIDACIÓN DEL DOCUMENTO

| ELABORADO POR: | REVISADO POR: | APROBADO POR: |
|---|---|---|
| Nombre: Johanny Valencia<br>Cargo: Gestor Desarrollo<br>Fecha: 17/05/22 | Nombre: Jasmin Herrera<br>Cargo: Gestor de calidad<br>Fecha: 16/06/22 | Nombre: Ingrid Solano<br>Cargo: Jefe OAI<br>Fecha: 16/06/22 |